CS 300 Pseudocode Document

Function Signatures

Below are the function signatures that you can fill in to address each of the three program requirements using each of the data structures. The pseudocode for printing course information, if a vector is the data structure, is also given to you below (depicted in bold).

```
// Vector pseudocode
// Reading Files
int readFile(string "filename") {
      Use stream to open file
            Open "fileName"
      WHILE (filename) {  //while there is something in the file
            For each line {
                  IF ( < 2 values in a line) {
                        RETURN ERROR
                  }
                  ELSE {
                        READ values
                        addCourse(values)
                  }
                  IF ( >= 3 values in line) {
                        IF (value is in a first value somewhere else) {
                              addCourse(values)
                        }
                        ELSE {
                              RETURN ERROR
                        }
                  }
            }
      }
}

// structure for Course information
struct CourseObj {
      INIT courseId
      INIT courseName
      INIT prerequisites
};

// store in data structure
Course addCourse(){
      while (!EOF){
            CourseObj course
            course.courseId = file[i][1]
            course.courseName = file[i][2]
            course.prerequisites = file[i][3]
            courseList.push_back(course)
      }
      Return course
```

```
}


int numPrerequisiteCourses(Vector<Course> courses, Course c) {
      totalPrerequisites = prerequisites of course c
      for each prerequisite p in totalPrerequisites
            add prerequisites of p to totalPrerequisites
      print number of totalPrerequisites
}

void printSampleSchedule(Vector<Course> courses) {
      for each item in vector
            print item (course)
}

void printCourseInformation(Vector<Course> courses, String
courseNumber) {
      for all courses
            if the course is the same as courseNumber
                  print out the course information
                  for each prerequisite of the course
                        print the prerequisite course information
}
```

```
// Hashtable pseudocode
// loading courses to hashtable
Void loadCourse(string "filename", HashTable* hashTable) {
      Use stream to open file
            Open "fileName"
      WHILE (filename) {  //while there is something in the file
            For each line {
                  course = addCourse()
                  insertHashData(course, hashTable)
      }
}

//Insert Data to hashtable
Void insertHashData(course, hashTable) {
      Create key for course
      IF (searchHashData(course->key, hashTable == NULL) {
            //hash() is not defined in this pseudocode
            bucketList = hashTable[hash(course->key)]
            node = new linked list node
            node->next = NULL
            node->data = course
            AppendToList(node)
      }
```

```
}

// search for data in hashtable
Void searchHashData(key, hashTable) {
      bucketList = hashTable[hash(key)]
      courseNode = search(key, bucketList)
      IF (courseNode != NULL) {
            RETURN courseNode->data
      }
      ELSE {
            RETURN NULL
      }
}
// create hashtable structure
Struct Node{
      Course course
      Int key
}
INIT Vector nodes
INIT int tableSize


void printSampleSchedule(Hashtable<Course> courses) {
      for each bucket {
            IF (i->key != head) {
                  PRINT the course information
                  Node* node = i->next
                  WHILE (node != NULL) {
                        PRINT the course information
                        node = node->next
                  }
            }
      }
}

void printCourseInformation(Hashtable<Course> courses, String
courseNumber) {
      searchedC = searchHashData()
      PRINT searchedC

}
```

// Tree pseudocode

```
//Binary Search Tree Object
BinarySearchTree* bst
Bst = new BinarySearchTree()

//inserting a node into a tree
void insertTreeNode(bst, node){
      IF (tree->root is NULL) {
            Tree->root = node
```

```
            Node->left = NULL
            Node->right = NULL
      }
      ELSE {
            current = tree->root
            WHILE (current IS NOT NULL){
                  IF (node->key < current->key) {
                        IF (current->left IS NULL){
                              current->left = node
                              current = NULL
                        }
                        ELSE {
                              current = current->left
                        }
                  }
                  ELSE {
                        IF (current->right IS NULL){
                              current->right = node
                              current = NULL
                        }
                        ELSE {
                              current = current->right
                        }
                  }
            }
            Node->left = NULL
            Node->right = NULL
      }
}

// loading courses to Binary Search Tree
Void loadCourse(string "filename", BinarySearchTree* bst) {
      Use stream to open file
            Open "fileName"
      WHILE (filename) {  //while there is something in the file
            For each line {
                  course = addCourse()
                  insertTreeNode(course, hashTable)
      }
}

//search Binary Search Tree
Void searchBST (bst, key) {
      Current = tree->root
      WHILE (current IS NOT NULL) {
            IF (key == current->key) {
                  RETURN current
            }
            ELSE IF (key < current->key) {
                  Current = current->left
            }
```

```
            ELSE {
                  Current = current->right
            }
      }

      Return NULL
}

void printSampleSchedule(Tree<Course> courses) {
      if (node IS NULL) {
            RETURN
      }
      printSampleSchedule (node->left)
      PRINT node
      printSampleSchedule (node->right)


}


void printCourseInformation(Tree<Course> courses, String
courseNumber){
      searchedC = searchBST(bst, courseNumber)
      PRINT searched C
}

//MENU Pseudocode
int main() {
      PRINT "1: Load course data"
      PRINT "2: Print course list"
      PRINT "3: Print single course"
      PRINT "9: Exit program"
      IF case "1":
            PRINT "Which data type"
            PRINT "1: vector, 2: hashtable, 3: Binary Search Tree"
            IF case "1":
                  CALL loadCourse("filename", vector)
            IF case "2":
                  CALL loadCourse("filename", hashTable)
            IF case "3":
                  CALL loadCourse("filename", BinarySearchTree)
      IF case "2":
            CALL printSampleSchedule()
      IF case "3":
            PRINT "What course do you want to see?"
            GET INPUT
            CALL printCourseInformation(input)
      IF case "9":
            EXIT
}
```

Example Runtime Analysis

When you are ready to begin analyzing the runtime for the data structures that you have created pseudocode for, use the chart below to support your work. This example is for printing course

information when using the vector data structure. As a reminder, this is the same pairing that was bolded in the pseudocode from the first part of this document.

Code	Line Cost	# Times Executes	Total Cost
`for all courses`	1	n	N
`    if the course is the same as courseNumber`	1	n	N
`        print out the course information`	1	1	1
`        for each prerequisite of the course`	1	n	N
`          print the prerequisite course information`	1	n	N
		Total Cost	4n + 1
		Runtime	O(n)

	Advantages	Disadvantages
Vector or Linked List		
Hash Table		
Binary Search Tree		